What you will build

Your task is to build a REST API that exposes methods to interact with a cache that you will also build.

- Add an endpoint that returns the cached data for a given key
    - If the key is not found in the cache:
        - Log an output “Cache miss”
        - Create a random string
        - Update the cache with this random string
        - Return the random string
    - If the key is found in the cache:
        - Log an output “Cache hit”
        - Get the data for this key
        - Return the data
- Add an endpoint that returns all stored keys in the cache
- Add an endpoint that creates and updates the data for a given key
- Add an endpoint that removes a given key from the cache
- Add an endpoint that removes all keys from the cache

Functional requirements

- All data returned by the cache is random dummy data.
- The cache does not have another data source in the background to be cached.
- The number of entries allowed in the cache is limited. If the maximum amount of cached items is reached, some old entry needs to be overwritten. Please explain the approach of what is overwritten in the comments of the source code.
- Every cached item has a Time To Live (TTL). If the TTL is exceeded, the cached data will not be used. A new random value will then be generated (just like cache miss). The TTL will be reset on every read/cache hit.
- You do not need to build a frontend, the API shall be used with tools like curl or Postman.

Technical requirements

- npm install must resolve all the dependencies.
- Standard npm commands must be implemented, “install”, “start” and “test”.
- Use MongoDB to store the cache data in.
- Use only Node.js and Express.js to build the API. Let us know what node version you used.
- It’s up to you what version of Javascript to use. You can use ES2016 or other tools like Typescript or Flow. It just needs to be able to run on the defined node version.
- The project has to work in a Unix environment.

Additional requirements for this challenge

- Create a public Github repo for the code.
- Commit your progress with good commit messages and in sensible chunks.
- The person that will review your code will only have Node, npm and MongoDb installed. Any other dependency must be solved by your npm scripts.

Have these considerations in mind

- Test the code in a way that seems appropriate to you in a professional environment.
- Think about the names of the endpoints, the HTTP methods to use (GET, PUT, POST, DELETE, etc.) and the structure of the returned data.
- Use the appropriate status codes (also in case of an error).
- Structure the code as if it was a huge application and make sure that the code will still be readable and maintainable when that point is reached.
- Pretend you have a big team and make it really easy for them to set up your environment.
- Think about what data should be configurable in different environments (e.g. db connection).

Happy coding :-D